SECTION ONE

STANDARD FORTRAN

FORTRAN

FORTRAN is short for FORmula TRANslation. FORTRAN is a computer language, that is, a language which people use to control computers. It was the first widely used computer language that was designed for people rather than for a machine. A FORTRAN program resembles English. Before FORTRAN, programming had to be done in machine code. A machine code program is just a series of numbers. For this reason, FORTRAN was called a "high level language." Today there are thousands of other high level languages, but FORTRAN remains the most common. BASIC is another popular high level language. Those of you familar with BASIC will notice many similarities between BASIC and FORTRAN.

To give you an idea of how a FORTRAN program looks, examine the following very simple example.

```
SIMPLE
1Ø    FORMAT(F1Ø.4)
      TOTAL=2.Ø÷5.Ø
      WRITE (3,1Ø) TOTAL
      STOP
      END
```

OMSI 2ØØØ FORTRAN is a series of computer programs written in the BASIC language which run short FORTRAN programs. To use OMSI 2ØØØ FORTRAN, you must first log on to your Hewlett-Packard 2ØØØ series computer in the normal fashion. If you do not know how to log on, you should either get help from your instructor, or consult an H-P 2ØØØ BASIC manual. After you have logged on, you must get the program 'FORTRN' from storage (GET-FORTRN) and run this program. If the system is not available, check an OMSI 2ØØØ FORTRAN manual, or get help from your instructor. If the system is available, after you have typed 'RUN', it will respond with 'OMSI 2ØØØ FORTRAN', and then a period on the left margin. You are now ready to start programming in FORTRAN.

FORTRAN PROGRAMS

On the previous page, a very simple FORTRAN program was introduced, and at this time, it should be more closely examined. The program was:

```
SIMPLE
1Ø    FORMAT(F1Ø.4)
      TOTAL=2.Ø+5.Ø
      WRITE (3,1Ø) TOTAL
      STOP
      END
```

A computer program is a set of instructions which tells the computer exactly what to do; this program tells the computer to find the sum of 2.Ø and 5.Ø, print the result out on the teleprinter, and stop. Although this program is very short, it shows several of the important rules that must be remembered for all FORTRAN programs. Each line of a FORTRAN program is a separate, distinct instruction. Each line is a single "step" or "statement".

Starting from the top, the first line is "SIMPLE". This is not really an instruction to the computer, just the name of the program. Every FORTRAN program must have a name. The next line is "1Ø FORMAT(F1Ø.4)". The "1Ø" at the start of the line is called the statement number; statement numbers are used to reference a line from within a program. The word "FORMAT" describes what kind of statement it is. What the FORMAT statement does and how it is used will be discussed later, but right now you must remember that ALL FORMAT STATEMENTS MUST BE AT THE FIRST OF THE PROGRAM, only the name can be before the FORMAT statements.

The next line, " TOTAL=2.Ø+5.Ø" is called an assignment statement. It assigns the numeric value of the expression (in this case, 2.Ø+5.Ø) on the right of the equals sign to the variable on the left side of the equals sign. In this case that variable is named 'TOTAL'. This will also be very thouroughly discussed later in the book. It is important to notice that the six spaces in front of the statement "TOTAL=2.Ø+5.Ø" are a part of it. In FORTRAN, THE ONLY THING ALLOWED IN THE FIRST SIX SPACES OF A LINE IS THE STATEMENT NUMBER, EVERYTHING ELSE MUST START IN COLUMN SEVEN. If a line does not have a statement number, these spaces must be left blank. You will notice that the word FORMAT in the FORMAT statement starts in the seventh column.

The fourth line is "   WRITE (3,1Ø) TOTAL". Once again, notice that the first six columns are blank, and that WRITE starts in the seventh column. This line is an instruction to the computer to print out the numeric value of the variable TOTAL on the teleprinter. "   STOP" tells the computer to stop running, that the program is done computing. The final line is "   END". This is an instruction to the computer telling that there is no more program instructions after that line. THE FINAL LINE OF EVERY PROGRAM MUST BE AN 'END' AND ONLY THE FINAL LINE CAN BE AN 'END'. Again the correct number of spaces must be used before the word 'END'.

For a program to run, the steps must be in the correct order. The computer does each step working from the top of the list down. This program is wrong.

```
WRONG1
1Ø    FORMAT (F1Ø.4)
      TOTAL=2.Ø÷5.Ø
      STOP
      WRITE (3,1Ø) TOTAL
      END
```

It will not work correctly because the computer is told to stop before it is told to print out the value of variable TOTAL. The program

```
WRONG2
1Ø    FORMAT (F1Ø.4)
      WRITE (3,1Ø) TOTAL
      TOTAL=2.Ø÷5.Ø
      STOP
      END
```

will not run correctly either since the computer is told to print out the value of the variable TOTAL before it is told what that value is.

We will talk more about variables, statements, and other matters further on in the book, but for the moment the important points to remember are the rules for spacing and ordering the statements in a FORTRAN program.

EXERCISES 1.2

1. What will these programs do?

a)
```
SIMPLE1
1Ø      FORMAT (F1Ø.4)
        TOTAL=3.Ø+3.Ø
        WRITE (3,1Ø) TOTAL
        STOP
        END
```

b)
```
SIMPLE2
1Ø      FORMAT (F1Ø.4)
        TOTAL=2.Ø+3.Ø
        STOP
        END
```

2. What is wrong with each of these programs?

a)
```
WRONG3
1Ø      FORMAT (F1Ø.4)
        TOTAL=2.Ø÷3.Ø
        WRITE (3,1Ø) TOTAL
        STOP
```

b)
```
WRONG4
1Ø      FORMAT (F1Ø.4)
TOTAL=2.Ø÷3.Ø
        WRITE (3,1Ø) TOTAL
        STOP
        END
```

c)
```
WRONG5
        TOTAL=2.Ø÷3.Ø
        WRITE (3,1Ø) TOTAL
1Ø      FORMAT (F1Ø.4)
        STOP
        END
```

OPERATING OMSI 2000 FORTRAN

Before one can run a FORTRAN program it is necessary to learn to operate OMSI 2000 FORTRAN system. When you have logged on to Hewlett Packard BASIC, typed GET-FORTRN and RUN, the following response should be printed out (in this book, all print out from the computer will be in bold face type):

```
FORTRN

OMSI 2000 FORTRAN

.
```

The period means you are in the "monitor", and OMSI 2000 FORTRAN is ready for your use. The user operates the system from the monitor, using a series of special commands. Commands are not part of a program; they are specific instructions that the user tells the computer to make it possible to run his program.

The first command that is needed to run a FORTRAN program is the LOAD command. This command is used to make the computer accept the user's program. The computer identifies commands by their first two letters, so LO means the same as LOAD to the monitor. When the user types LOAD and presses the return key (the return key must be pressed after everything typed into the computer), it will type out:

```
.LOAD
PROGRAM NAME--
```

Now the computer is ready to take the program into storage. It will not do any of the instructions in the program now. LOAD just gets the FORTRAN program into the computer so that it can be run at a later time. After the user types in the program name, the text of the program can be typed in. For example, the program SIMPLE would be loaded like this:

```
.LOAD
PROGRAM NAME--SIMPLE
10     FORMAT (F10.4)
```

followed by the rest of the program. As has been mentioned previously, the first six spaces in each line of a FORTRAN program are reserved for a line number. OMSI 2000 FORTRAN has a special character to "tab" over to the seventh space, where the rest of the line must begin. The character used is control I. Control I is made by holding down the "control" key while typing 'I'. In the rest of the book, control I is represented by "I^C". Tabbing the program SIMPLE to the correct spacing would be done like this:

```
1ØI^C FORMAT (F1Ø.4)
I^C TOTAL=2.Ø+5.Ø
I^C WRITE(3,1Ø) TOTAL
I^C STOP
I^C END
```

The spacing would be correct since "I^C STOP" is the same as " STOP" and "1ØI^C FORMAT (F1Ø.4)"is the same as "1Ø FORMAT (F1Ø.4)". The user will not "see" the line move over, but it means the same thing. Now the entire program SIMPLE can be loaded.

```
.LO

PROGRAM NAME--SIMPLE
1ØI^C FORMAT (F1Ø.4)
I^C TOTAL=2.Ø+5.Ø
I^C WRITE(3,1Ø) TOTAL
I^C STOP
I^C END
```

Now the text has been loaded and the user must return to the monitor to type any further commands. Typing a control Y (Y^C) will exit the user from the LOAD command. Having completed loading SIMPLE, the Y^C will return the user to the monitor:

```
Y^C

^Y

EXIT

.
```

This period indicates that the computer is ready to accept further commands

To have the computer "execute" a FORTRAN program that has been "LOADed", the EXECUTE command is used. EXECUTE tells the computer to do the instructions in the program. Having loaded the program SIMPLE, typing EXECUTE will run it, resulting in the following print out:

```
.EX

LOADING    FORTRAN 'SIMPLE'

EXECUTION  FORTRAN 'SIMPLE'

    7.ØØØØ

EXECUTION COMPLETED
```

The 7.ØØØØ is the result, and the STOP brought the computer back to the monitor mode so that additional commands can now be typed in. If there had been any errors in the program, the computer would have printed them out after the user had typed EXECUTE. For example, if the word FORMAT had been spelled FROMAT, the following would have been printed out:

```
.EX

LOADING    FORTRAN 'SIMPLE'
1Ø     FROMAT (F1Ø.4)
ILLEGAL OPERAND
EXECUTION ABORTED
.
```

To get a listing of the current FORTRAN program the LIST command is used. A listing of the program which contained the error above could be obtained by typing LIST after the computer printed out the period. The user can now see that I^{c} has correctly spaced the program.

```
.LIST

SIMPLE
1Ø     FROMAT (F1Ø.4)
       TOTAL=2.Ø+5.Ø
       WRITE(3,1Ø) TOTAL
       STOP
       END
```

When the user is done, he can return to BASIC by the UJOB command. UJOB will not erase the current FORTRAN program.

```
.UJOB

LOG OUT:  FORTRAN
PLEASE REPORT MALFUNCTIONS OR COMMENTS TO THE SOFTWARE DEVELOPMENT
GROUP AT THE OREGON MUSEUM OF SCIENCE AND INDUSTRY, PORTLAND
OREGON, 97221
```

It is very important to become familiar with these commands. Without the ability to operate the OMSI 2ØØØ FORTRAN system, it is impossible to run FORTRAN programs.

EXERCISES 1.3

1. What will the following commands do when typed in to the monitor after it has printed a period?

a) EXECUTE
b) RUN
c) Y^C
d) FORMAT
e) LIST
f) UJOB
g) LODE
h) I^C

2. LOAD, EXECUTE, and LIST the FORTRAN program SIMPLE.

3. Check your answers to problems 1a and 1b from exercises 1.2 by running those programs.

ASSIGNMENT STATEMENTS

Assignment statements are used to assign a value to a variable. In FORTRAN, a variable is a storage location in the machine, not an algebraic variable. For that reason, an expression that would not make any sense in algebra, "A=A+1", is perfectly acceptable in FORTRAN. Assigning a value to a variable is sometimes called "defining" the variable.

There are rules for variable names in FORTRAN that must be remembered for successful programming. First, a variable name may have one to six characters, and the first character must be a letter. The rest of the characters may be either letters or digits. Some examples of legal variable names are INFLAT, PLINTH, ISINK, and BOZO23. Some illegal variables are SCHWARTZ (too many characters), MAN#5 (# is not a letter or digit), and 2INTEG (it does not start with a letter).

FORTRAN has two kinds (called "modes") of variables, floating point and integer. An integer variable must have a whole number value. Floating point variables may have any real value (for this reason they are often called "real" variables). Any variable with the first letter I, J, K, L, M, or N is automatically an integer variable. Variables starting with any other letter are automatically floating point. For example, ZAPP is a floating point variable; IMP is an integer variable.

Expressions in FORTRAN also have modes. An expression is a correctly ordered series of values and operators, such as:

```
2.Ø+2.Ø          or     4*2+1        or
(3.Ø+5.Ø)/7.5    or     (2+3)**4
```

In FORTRAN the symbol "+" is used for addition. The symbol "-" is used for subtraction. Multiplication is represented by "*" and division by "/". Expontentiation is given by the symbol "**" (two to the third power, 2^3 ,is written 2**3) Expressions are evaluated by a standard procedure. The computer does all expontentiation first, then all multiplication and division (if there is more than one "*" or "/" in an expression, the computer works from left to right). Then the computer does all additions and subtractions, working from left to right. Any quantity placed in parantheses will be evaluated first. For example:

```
1+2*4=1+(2*4)=9
(1+2)*4=12

2.Ø+2.Ø**3.Ø=2.Ø+(2.Ø**3.Ø)=1Ø.Ø
(2.Ø+2.Ø)**3.Ø=64.Ø

8+4**2/2=8+((4**2)/2)=16
(8+4)**2/2=((8+4)**2)/2=72
8+4**(2/2)=8+(4**(2/2))=12
```

An expression like "1+2*4" is an integer expression since it contains only integral quantities, such as "4". The expression "2.Ø+2.Ø**3.Ø" is a floating point (real) expression since it contains quantities that have decimal points. All floating point constants must have a decimal point, and integer quantities must not have decimal points. Within a single expression ALL QUANTITIES MUST HAVE THE SAME MODE. Expressions like "2.Ø*2" or "4+3.5" are not acceptable FORTRAN expression because they contain more than one mode.

Variables can also be used in expressions, as long as they are of the same mode as the rest of expression and have been defined previously in the program. Some acceptable examples are:

```
SAM+2.5
(INDEX+7)/2
2*(INK+LINK+MINK)-KINK/3
PEGLEG**2.Ø+CROCK**2.Ø
```

As mentioned earlier, the purpose of the assignment statement is to assign values to variables. The general format for this is placing the variable that is to be assigned on the left side of an equal sign and the expression on the right side of the equal sign, like this:

```
INK=1+2*4
ALPHA=(2.Ø+2.Ø)**3.Ø
REGLEN=PEGLEG**2.Ø+CROCK**2.Ø
KOUNT=KOUNT+1
```

The expression "INK=1+2*4" will store the value of "1+2*4" in the location (variable) "INK". The expression "KOUNT=KOUNT+1" means add 1 to the value in location (variable) KOUNT and store this new value in location (variable) KOUNT.

Expressions always return a value in the same mode as the expresssion. If the line "IVALUE=7/2" was executed in a program, IVALUE would have value 3. The line "IVALUE=-7/2" would return the value -3 to IVALUE. In both cases, since both values are integer constants, their quotient was simply truncated. The mode of the variable assigned should be the same as the expression.

By changing the expression in SIMPLE, the program can be used to print out the value of any numeric expression. For example, if (2.5+3.4)/7.1+12.45**2.Ø needed to be evaluated, SIMPLE would be rewritten

```
SIMPLE3
1Ø    FORMAT (F1Ø.4)
      TOTAL=(2.5+3.4)/7.1+12.45**2.Ø
      WRITE (3,1Ø) TOTAL
      STOP
      END
```

Execution of this program will output " 155.8335", the value of this expression.

FORTRAN has available a series of numeric valued functions for use in expressions. Available functions include

SIN(rx)	sine of an angle in radians
COS(rx)	cosine of an angle in radians
AINT(rx)	truncates digit after decimal point
SQRT(rx)	square root
ABS(rx)	absolute value

All the functions shown must have floating point arguments and all will return floating point values. A complete list all the functions available can be found in an OMSI 2ØØØ FORTRAN manual. If a program to find the sine of a value was needed, SIMPLE could be modified.

```
SIMPLE4
1Ø    FORMAT (F1Ø.4)
      ALPHA=3.14159/4.Ø
      TOTAL=SIN(ALPHA)
      WRITE(3,1Ø) TOTAL
      STOP
      END
```

When executed the results are

```
.EX
LOADING   FORTRAN 'SIMPLE4'
EXECUTION FORTRAN 'SIMPLE4'
     .7Ø71
EXECUTION COMPLETED
.
```

Two very important functions are FLOAT and IFIX. These functions change mode. FLOAT will change an integer value into a floating point value. IFIX will change a floating point value into an integer. For eample, if the values of TOTAL and IMP needed to be added together, the mode of one would have to be changed. This could be done two ways

```
IMPTOT=IMP+IFIX(TOTAL)
          or
TOTIMP=FLOAT(IMP)+TOTAL
```

The method chosen would depend on what mode the result was to be.

The assignment statement is the heart of a program. FORTRAN is a FORmula TRANslater because it can "translate" complicated assignment statements into machine operations.

EXERCISES 1.4

1) Which of these variables are legal, and if they are legal, what mode are they?

a) ALPHA b) OUTPUT c) JAM
d) SOCK12 e) 12MORE f) HOT-45

2) Which of the folowing expressions are legal, and what is the mode of the legal expressions?

a) APA+2.3 b) 2*(IMP+1.Ø) c) 2.Ø**3.Ø+THETA15
d) (2**(JOG*(3+J))) e) 2.Ø*(G+H+I+J)/3 f) (IMP+(2*INK)

3) What is the value of these FORTRAN expressions, given that ALPHA=2.5, BETA=5.Ø, IMP=4, INK=3, and JOGGER=1?

a) 5/2*2 b) 2.Ø**3.Ø+1Ø.Ø/2.5-2.Ø*4.5
c) ALPHA+BETA d) (INK+2)*IMP
e) BETA**2.Ø+ALPHA*2.Ø f) JOGGER+IMP*INK

4) Rewrite SIMPLE to find the value of the following expressions.
(NOTE: Remember that TOTAL is a REAL variable).

a) The product of 5 and 6
b) Expression b in exercise 3.
c) The square root of pi (3.1415926)
d) Two squared plus three squared plus four squared.

5) Write a set of FORTRAN expressions that will set the variable BETA to be the square of the variable ALPHA, the variable DELTA to be the cube of ALPHA, and the variable TOTAL to be the sum of BETA and DELTA. Using the program SIMPLE as a guide for outputting TOTAL, set ALPHA to be 3, and run this program.

6) Write a FORTRAN expression for the discriminant of the quadratic equation ($d=b^2-4ac$, for the equation $ax^2+bx+c=Ø$). Using this expression, write a FORTRAN program to find the discriminant of $4x^2-5x+6=Ø$.

To output information from the computer to the user, a combination of two FORTRAN statements is needed, WRITE and FORMAT. The WRITE statement tells the computer to print out the values of a list of variables; the FORMAT statement tells exactly where the numbers are to be placed.

The WRITE statement is the simpler of the two. It has the general form:

WRITE (3,<FORMAT reference>) <variable list>

The 3 is the device code. The device code specifies the device on which output will be printed. In OMSI 2ØØØ FORTRAN the only device available is the teleprinter, so 3 is the only device code. "<FORMAT reference>" is the line number of the FORMAT statement that the WRITE will use to position the output on the page. "<variable list>" is a list of variables that the program will print out. So the statement

```
WRITE (3,1Ø) INDEX,IVALUE
```

will print out the values of the variables INDEX and IVALUE out on the teleprinter using the FORMAT statement with line number 1Ø. The statement

```
WRITE (3,2ØØ) A,SQRA,CUBA
```

will print out the values of A, SQRA, and CUBA on the teleprinter using the FORMAT statement with line number 2ØØ.

FORMAT statements are trickier. It is important to remember that the user has to specify what he wants in every space of every line. A line on a teleprinter has 72 spaces. The program SIMPLE uses a very simple WRITE-FORMAT combination:

```
1Ø    FORMAT (F1Ø.4)
      ---
      WRITE (3,1Ø) TOTAL
```

The value of the variable total will be printed out using FORMAT statement "1Ø". That FORMAT has the code F1Ø.4 as the output specification. "F1Ø.4" reserves the first ten spaces (it is the first specification) for a floating point value. The F stands for floating point. The 4 tells the computer to print 4 digits after the decimal point. The number will be printed as far to left in the field as possible. The first unused spaces are filled with blank spaces. The program SIMPLE printed out the following number field:

```
  7.ØØØØ
```

It used a 1Ø space field and printed 4 digits to the left of the decimal point, even though these digits were all zeroes.

If the format specification on SIMPLE was changed to F1Ø.3 the ouput would be

7.ØØØ

If the FORMAT specification was changed to F9.5, the output would then be

7.ØØØØØ

In selecting FORMAT specifications, the user must be careful to allow enough space in the field for a decimal point and for a negative sign (in case the value is less than zero). The specification F1.Ø is not acceptable because it reserves only enough space for the decimal point, leaving no room for any digits. F2.Ø will not be acceptable for a negative number because it only has space for the sign and the decimal point, again leaving no room for any digits. The use of a field that is too small for the number, such as printing 1ØØØ.Ø2 using a F5.2 specification, will cause errors in the output.

The F (floating point) format specification is for floating point numbers so variables printed using F format must be floating point variables. For integer variables, the I format specification is used. It has the form "Iw", where w is the width of the field. Since integer variables can not have decimal points, there is no need to reserve space for digits after the decimal point. The value of the variable is printed in the spaces furthest to the right, with the first, unused spaces being left blank. The program

```
INTEGER
3Ø    FORMAT (I7)
      IN=2Ø+2ØØ
      WRITE(3,3Ø) IN
      STOP
      END
```

will give the following output when executed.

.EX

LOADING FORTRAN 'INTEGER'

EXECUTION FORTRAN 'INTEGER'

22Ø

If the format specification I5 had been used in the program INTEGER, the results would have been printed out like this:

```
  22Ø
```

WRITE and FORMAT can be used to print out more than one value at a time by seperating the FORMAT specifications by commas and matching them to the variable list on the WRITE statement. The program

```
SQUARES
1Ø    FORMAT (I7,F1Ø.3)
      FULL=3.Ø**2.Ø
      IMFULL=4**2
      WRITE (3,1Ø) IMFULL,FULL
      STOP
      END
```

causes the following printout when executed.

```
.EXECUTE

LOADING    FORTRAN 'SQUARES'

EXECUTION  FORTRAN 'SQUARES'

     16      9.ØØØ

EXECUTION COMPLETED
.
```

The first variable, IMFULL is printed using the I7 specification. The value of IMFULL appears in the two left most spaces of that field with the first five spaces filled with blanks. FULL, the second variable in the list, is printed through the second specificaion, F1Ø.3. Its value appears in spaces 8 through 17 of the line. Note that the mode of the variable and the mode of the FORMAT specification are carefully matched. If the program had been written as follows, it would not have run.

```
.LIST
SQUARES
1Ø      FORMAT (F1Ø.3,I7)
        FULL=3.Ø**2.Ø
        IMFULL=4**2
        WRITE (3,1Ø) IMFULL,FULL
        STOP
        END
.EXECUTE

LOADING     FORTRAN 'SQUARES'

EXECUTION   FORTRAN 'SQUARES'

        WRITE (3,1Ø) IMFULL,FULL
!!!!!! MIXED MODE !!!!!!
EXECUTION TERMINATED
.
```

This is clearly not the desired output.

To print headings, titles, or messages, the user simply encloses the desired text in apostrophes and puts them in the appropriate place in the FORMAT specifications. For example, "1Ø FORMAT (' HOWDY')" will cause the word HOWDY to be printed out on the teleprinter when it is referenced by a WRITE statement. ' HOWDY' is called a literal field. If you are on a HP 2ØØØC or 2ØØØF computer, you should leave the first character of the first string a blank space; OMSI 2ØØØC/F FORTRAN uses this character to specify carriage control. A blank will cause normal, single spacing. For more information on carriage control, see the 2ØØØC/F section of the OMSI 2ØØØ FORTRAN manual.

A simple example of the use of a literal field is the program HEADER.

```
HEADER
1Ø      FORMAT (' THIS IS A SAMPLE OF A LITERAL FIELD')
        WRITE (3,1Ø)
        STOP
        END
```

If HEADER is executed, the following is printed out.

```
.EX

LOADING     FORTRAN 'HEADER'

EXECUTION   FORTRAN 'HEADER'


THIS IS A SAMPLE OF A LITERAL FIELD


EXECUTION COMPLETED
.
```

Note that there is no variable list on the WRITE statement in HEADER, since no numeric values can be printed out through a literal field.

Literal fields are often used with other FORMAT specifications to help clarify the output. The program SQUARES could be rewritten using literals so that the output is identified.

```
SQUARES
1Ø      FORMAT (' FOUR SQUARED =',I7,'    THREE SQUARED=',F1Ø.3)
        FULL=3.Ø**2.Ø
        IMFULL=4**2
        WRITE (3,1Ø) IMFULL,FULL
        STOP
        END
```

Now, when SQUARES is executed, the output is:

```
.EX

LOADING     FORTRAN 'SQUARES'

EXECUTION   FORTRAN 'SQUARES'


FOUR SQUARED=      16    THREE SQUARED=      9.ØØØ

EXECUTION COMPLETED
.
```

Note that in SQUARES the first variable is printed out using the first numeric specification in the list, and the modes of the variables and the specifications are again carefully matched.

More than one FORMAT statement can be used in a FORTRAN program. Changing SQUARES to have two FORMAT statements (and two WRITE statements) changes the print out.

```
SQUARES
1Ø    FORMAT (' FOUR SUARED=',I7)
2Ø    FORMAT (' THREE SQUARED=',F1Ø.3)
      FULL=3.Ø**2.Ø
      IMFULL=4**2
      WRITE (3,2Ø) FULL
      WRITE (3,1Ø) IMFULL
      STOP
      END
```

```
.EX

LOADING   FORTRAN 'SQUARES'

EXECUTION FORTRAN 'SQUARES'


THREE SQUARED=      9.ØØØ
FOUR SQUARED=      16

EXECUTION COMPLETED
.
```

One of the rules for FORTRAN program construction was that all FORMAT statements must occur at the first of a program. This rule can now be explained. A FORMAT statement is not a direct instrucion to the computer; it must be referenced by another statement. For this reason it is called a non-executable statement. All the other statements discussed so far have been "executable" statements. A FORMAT statement, not being a direct instruction to the computer cannot occur in middle of a group of "executable" statements. If a FORMAT statement did occur there the computer would not know what to do with it. For that reason, all non-executable statements (FORMAT is the only one so far) must be first in a FORTRAN program. Execution begins at the first executable statement.

As the reader can clearly see, there are many ways to FORMAT any single output. It is the programmer's job to make output precise and clear.

EXERICSES 1.5

1) What will the following programs print out when they are run? Run one and test yourself.

a) ?OUTPUT1?

```
1Ø    FORMAT(' THE SUM OF',I3,' AND',I3,' IS',I4)
      IMP=23
      INK=45
      JOGGER=IMP+INK
      WRITE(3,1Ø) IMP,INK,JOGGER
      STOP
      END
```

b) ?OUTPUT2?

```
1Ø    FORMAT(F6.1,F6.2,F6.3,I5)
      ALPHA=173.7
      BETA=1.11
      GAMMA=ALPHA*BETA
      ICK=2ØØØ+2ØØ+2Ø+2
      WRITE(3,1Ø) ALPHA,BETA,GAMMA,ICK
      STOP
      END
```

c) ?OUTPUT3?

```
15Ø   FORMAT(I5,I5,I1Ø)
      INK=1Ø1Ø
      IMP=2Ø2Ø
      IBOBO=INK*IMP
      WRITE(3,15Ø) INK,IMP,IBOBO
      STOP
      END
```

2) Write FORMAT specifications that will print out the following lines (# means a digit).

a) ####.##
b) FOUR TIMES FIVE = ###
c) ####
d) #####. /5 = ##.####
e) ## PLUS ## = ####
f) HOWDY!!!

3) Write a program which will identify and print out the sum, difference, product and quotient of 5.75 and 8.5Ø. Check your program by running it.

4) Write a program that will tell the number of ounces in a number of gallons of liquid. The variable IGAL should be set to be the number of gallons of liquid, and have the program print out the number of gallons and ounces and identify both.
(Hint: 1 gallon = 8 pints, and 1 pint = 16 ounces)

5) Debug these programs:

a) FIXIT1

```
1Ø    FORMAT(I5,F7.2)
      JOG=35*7
      ZILCH=63.Ø/9.Ø+155Ø.4
      WRITE(3,1Ø) ZILCH,JOG
      STOP
      END
```

b) FIXIT2

```
1Ø    FORMAT(F7,I7.Ø,' ARE THE ANSWERS.)
      INDIA=1Ø1*22
      AFGAN=37.2+98/.55
      WRITE(3,1Ø) INDIA,AFGAN
      STOP
      END
```

Until now, all variables have been defined by an assignment statement. This method works fine, but the program has to be rewritten for each different set of data. Using a READ statement, the user can give different values to variables while the program is running.

READ statements, like WRITE statements, must use a FORMAT statement. The FORMAT with a WRITE statement specifies what will be printed in every space of the line of output. With READ statements, the FORMAT specifies what is to be found in each space of a line of data.

The READ statements has two forms. Data can either be stored at the end of the program (after the END statement) or may be typed in when the READ is executed. Hence there are two device codes for READ. A statement like:

```
READ(1,1Ø) IVALUE
```

has device code "1". This code means that input from the teleprinter will be used to get a value for IVALUE. The line:

```
READ(2,1Ø) SPUD
```

has device code "2". This code is means a value for SPUD will be read from data stored at the end of the program. Like WRITE, lists of variables may be used in a READ.

The format specifications used with a READ are the same as those used with a WRITE. Floating point (real) variables still require F specifications, and Integer variables require I formatting. FORMAT specifications now describe the field where the value of the matching READ variable is to be found. The FORMAT

```
1Ø    FORMAT(F1Ø.3,I7)
```

specifies that in the first ten spaces one floating point value will be found, and in the next seven spaces, one integer value will be found. Matching this FORMAT to

```
READ(1,1Ø) SPUD,IDAHO
```

will store the first value (which is real) in SPUD, and the second value (which is an integer) in IDAHO. With device code 1, these values will be taken from a line of input data taken from the teleprinter.

The program SIMPLE can be modified using the READ statment so that it will add any two numbers.

```
SIMPLE5
1Ø     FORMAT(F7.2,F7.2)
2Ø     FORMAT(F7.2,' +',F7.2,'=',F8.2)
       READ(1,1Ø) VALU1,VALU2
       TOTAL=VALU1+VALU2
       WRITE(3,2Ø) VALU1,VALU2,TOTAL
       STOP
       END
```

The program is now designed to get two values from the teleprinter, add them, and print out their sum. When the program is run, the computer prints the following:

```
.EXECUTE

LOADING    FORTRAN 'SIMPLE5'

EXECUTION  FORTRAN 'SIMPLE5'


*
```

The asterisk indicates that the program is ready to accept input data through a READ statement. The input data must match the FORMAT specifications, so the value of VALU1 must be in the first seven spaces, and the value of VALU2 must be in spaces eight through fourteen. Two important points must be remembered for these data lines. First, any spaces are treated like zeroes. Secondly, the decimal point can occur anywhere in the field, but it must occur somewhere.

If the program is to add 7.Ø and 14.Ø, the input data line must have 7. in the first seven spaces, and 14. in the second seven spaces, like this:

```
*ØØØØ7.ØØØØ14.Ø
   7.ØØ+  14.ØØ=    21.ØØ

EXECUTION COMPLETED

.
```

or, the data line 'ØØ7.ØØØØØ14.ØØ' could be used just as well. If the progam is to be changed to use program data, it would look like this:

```
.LIST
SIMPLE6
1Ø      FORMAT(F7.2,F7.2)
2Ø      FORMAT(F7.2,' +',F7.2,'=',F8.2)
        READ(2,1Ø) VALU1,VALU2
        TOTAL=VALU1+VALU2
        WRITE(3,2Ø) TOTAL
        STOP
        END
ØØØ7.ØØØØ14.ØØ
```

.

The data to be read by the program is now stored after the END statement. The results are the same:

```
.EXECUTE

LOADING    FORTRAN 'SIMPLE6'

EXECUTION  FORTRAN 'SIMPLE6'

   7.ØØ+  14.ØØ=    21.ØØ

EXECUTION COMPLETED
```

.

To READ integer variables, the I format is used. Again, spaces are regarded as zeroes, so the user must be careful in positioning the number in the field. Suppose the following FORMAT statement was being used to read in values for the integer variables J1 and J2:

```
1Ø    FORMAT(I6,I5)
```

If the user wanted J1 and J2 to have values 16 and 8, respectively, then the data line would have to be:

```
ØØØØ16ØØØØ8
```

The number in the I6 field is ØØØØ16 or 16; the number in the I5 field is ØØØØ8, or 8.

If the data line "ØØØØ16ØØØ8" was used, the program might malfunction. The number in the I6 field is "ØØØØ16", or 16, but the number in the I5 field is "ØØØ8 ", or "ØØØ8Ø", which is 8Ø, not 8. A good way to avoid this is to use zeroes to fill in data lines. If you are on a 2ØØØB or 2ØØØE, you must uses zeroes to fill in blank spaces, since all leading spaces are deleted.

EXERCISES 1.6

1) Write correct input data lines for each of the following programs, and then tell what the computer's output will be. Run one of them to test yourself.

a.
```
INPUT1?
1Ø      FORMAT (I7,I7,I7)
2Ø      FORMAT (' THE SUM OF YOUR THREE NUMBERS IS',I8)
        READ (1,1Ø) INP1,INP2,INP3
        INPTOT=INP1+INP2+INP3
        WRITE (3,2Ø) INPTOT
        STOP
        END
```

b)
```
INPUT2?
1Ø      FORMAT (F1Ø.3,F1Ø.3)
2Ø      FORMAT (' THE QUOTIENT OF YOUR TWO NUMBERS IS',F11.5)
        READ (1,1Ø) DIVEND,DIVISO
        QUOTIT=DIVEND/DIVISO
        WRITE (3,2Ø) QUOTIT
        STOP
        END
```

c)
```
INPUT3
1Ø      FORMAT (I6,I8,F1Ø.3,F12.6)
2Ø      FORMAT (I12,F12.5)
        READ (1,1Ø) KIPP1,KIPP2,FISH1,FISH2
        KIPPER=KIPP1-KIPP1/KIPP2*KIPP2
        FISHER=FISH1-FISH1/FISH2*FISH2
        WRITE (3,2Ø) KIPPER,FISHER
        STOP
        END
```

2) Test another of your data lines from the first problem by inserting it as program data and changing the device code to read data.

3) Write data lines which when used with the given READ-FORMAT statements will set the variabls FROG, ICEPIK and SAM to the values 1.Ø, 193Ø, and 134.45, respectively.

a)
```
1Ø      FORMAT(F6.2,I1Ø,F5.1)
        READ(2,1Ø) SAM,ICEPIK,FROG
```

b)
```
29Ø     FORMAT(I5,F4.Ø,F1Ø.3)
        READ(2,29Ø) ICEPIK,FROG,SAM
```

c)
```
8Ø5     FORMAT(F1Ø.3,F7.3,I1Ø)
        READ(2,8Ø5) FROG,SAM,ICEPIK
```

4) Write a program to find the quotient and remainder of two values input from the terminal.

Control statements are instructions that cause the computer to skip or repeat other sections of the program. They "transfer control" to another part of the program. All control statements reference other lines by their line numbers. If a statment doesn't have a line number, it cannot be referenced. The simplest control statement is GO TO. It has the form "GO TO ####", where #### is the line number of another statement in the program. When the computer reaches a " GO TO 232" statement, instead of doing the next line after the GO TO 232 statement, it starts doing the statement with line number 232, and works the program from there. A simple program will show how this works.

```
TRANSFER
1Ø      FORMAT(' JIMMY CAN')
15      FORMAT(' NOT')
2Ø      FORMAT(' RUN')
        WRITE(3,1Ø)
        WRITE(3,15)
4       WRITE(3,2Ø)
        STOP
        END
```

When the program is run, the computer types out:

```
JIMMY CAN
NOT
RUN
```

However, if a GO TO statement is inserted, the results are quite different:

```
TRANSFER
1Ø      FORMAT(' JIMMY CAN')
15      FORMAT(' NOT')
2Ø      FORMAT(' RUN')
        WRITE(3,1Ø)
        GOTO 4
        WRITE(3,15)
4       WRITE(3,2Ø)
        STOP
        END
```

Now, when the program is run, the output is:

```
JIMMY CAN
RUN
```

The GO TO 4 statement caused the computer to skip over the WRITE(3,15) statement, and the program no longer executes this statement.

The GO TO statement is often used to cause the computer to repeat a portion of the program. If the user wanted the computer to print out the first 25 squares, he could write:

```
SQUARES
1Ø      FORMAT(I3' SQUARED =',I1Ø)
        I1=1
        I2=I1**2
        WRITE(3,1Ø) I1,I2
        I1=2
        I2=I1**2
        WRITE(3,1Ø) I1,I2
        I1=3
        I2=I1**2
        WRITE(3,1Ø) I1,I2
        I1=4
        I2=I1**2
        WRITE(3,1Ø) I1,I2
        I1=5
        I2=I1**2
        WRITE(3,1Ø) I1,I2
        I1=6
        I2=I1**2
        WRITE(3,1Ø) I1,I2
        I1=7
        I2=I1**2
        WRITE(3,1Ø) I1,I2
        I1=8
        I2=I1**2
        WRITE(3,1Ø) I1,I2
```

and so on, until I1=25, and then

```
        STOP
        END
```

Using a GO TO statement, a much shorter program can be used, repeating the statements that are the same on each group, a program like:

```
SQUARED
1Ø      FORMAT(I3,' SQUARED =',I1Ø)
        I1=Ø
2Ø      I1=I1+1
        I2=I1**2
        WRITE(3,1Ø) I1,I2
        GOTO 2Ø
        STOP
        END
```

This program is much shorter and it will print out as many squares as the user wants. In fact, it will continue to print squares until someone "unplugs" the computer. This program will never reach the STOP statement, so it will never stop itself. The only way to stop this program is to press the 'BREAK' button, and then GET-FORTRN and start over.

To allow limited repetition of parts of a program, the IF statement is used. The GOTO statement is an unconditional transfer, since it will always transfer control to the line referenced. IF is a conditional transfer which GOes TO different parts of the program depending upon the value of an expression which the IF statement contains. IF statements look like this:

```
IF(ALPHA-1.Ø) 1Ø,2Ø,3Ø
```

In general the form is

```
IF(<expression>) ##1,##2,##3
```

The expression is enclosed in parentheses and followed by three line numbers which are separated by commas. If the value of <expression> is less than zero, control is transfered to the first line number (##1). If the value of <expression> is zero, control is transfered to the second line number (##2), and if the value is greater than zero, control is transfered to the third line number (##3 This is easier to understand when seen in a program.

```
SIGN
1Ø    FORMAT(F1Ø.3)
2Ø    FORMAT(' VALUE IS NEGATIVE')
3Ø    FORMAT(' VALUE IS ZERO')
4Ø    FORMAT(' VALUE IS POSITIVE')
5Ø    READ(1,1Ø) VALUE
      IF(VALUE) 1ØØ,2ØØ,3ØØ
1ØØ   WRITE(3,2Ø)
      GOTO 5Ø
2ØØ   WRITE(3,3Ø)
      STOP
3ØØ   WRITE(3,4Ø)
      GOTO 5Ø
      END
```

This program will return the sign of a number read from teleprinter input until the user types a zero, after which it will stop. This way a large number of pieces of data can be processed without the program infinite looping.

The program SQUARES can also be modified using an IF statement so that it will stop by itself. If the first 25 squares are wanted the the variable I1 should not be greater than 25 which means that I1-25 must always be less than zero. Using this, the program could be written:

```
SQUARES
1Ø    FORMAT(I3,' SQUARED =',I1Ø)
      I1=Ø
2Ø    I1=I1+1
      I2=I1**2
      WRITE(3,1Ø) I1,I2
      IF (I1-25) 2Ø,3Ø,3Ø
3Ø    STOP
      END
```

When this program is run, the desired table of squares is printed.

EXERCISES 1.7

1. What will be the output from the following programs. Run one to test your answers.

a)
```
JUMP
1Ø     FORMAT (' COMPUTER AT LINE 13Ø')
2Ø     FORMAT (' COMPUTER AT LINE 15Ø')
3Ø     FORMAT (' COMPUTER IS ABOUT TO STOP.')
13Ø    WRITE (3,1Ø)
       GOTO 17Ø
15Ø    WRITE (3,2Ø)
17Ø    WRITE (3,3Ø)
       STOP
       END
```

b)
```
BRANCH
1Ø     FORMAT(I3)
2Ø     FORMAT(I3,'  IS EVEN')
3Ø     FORMAT(I3,'  IS ODD')
15     READ(1,1Ø) IPUT
       IF(IPUT) 25,7Ø,25
25     IF(IPUT-2*(IPUT/2)) 5Ø,4Ø,5Ø
4Ø     WRITE(3,2Ø) IPUT
       GOTO 15
5Ø     WRITE(3,3Ø) IPUT
       GOTO 15
7Ø     STOP
       END
```

c)
```
TABLE4
1Ø     FORMAT(' TABLE OF FOURTH POWERS')
2Ø     FORMAT('   X                  X4 ')
3Ø     FORMAT(I4,'              ',I7)
       I1=Ø
       WRITE(3,1Ø)
       WRITE(3,2Ø)
1ØØ    IF(I1-25) 12Ø,12Ø,13Ø
12Ø    I1=I1+1
       I4=I1**4
       WRITE(3,3Ø) I1,I4
       GOTO 1ØØ
13Ø    STOP
       END
```

2) Insert an IF statement in the following program so that it will tell the user he is trying to divide by zero, and then stop. Run this modified program to see if it works.

```
NAMELESS
1Ø      FORMAT(F1Ø.4,F1Ø.4)
2Ø      FORMAT(F15.6)
3Ø      FORMAT(' DIVISION BY ZERO')
4Ø      READ(1,1Ø) DIV1,DIV2
45      DIV3=DIV1/DIV2
        WRITE(3,2Ø) DIV3
        GOTO 4Ø
5Ø      WRITE(3,3Ø)
        STOP
        END
```

3) Write a program patterened after TABLE4 (problem 1c) to print out a table of the first 15 integers and their cubes.

4) Write a program to tell whether a number is a multiple of five. The number should be read from the teleprinter and if this number is zero, the program should stop.

DO LOOPS

In the last section, the IF statement was used to "LOOP" through a segment of the program SQUARES. The DO statement is used to loop through segments of programs in a similar manner. Using a DO statement, the program SQUARES would look like this:

```
SQUARES
1Ø    FORMAT(I3,' SQUARED =',I1Ø)
      DO 3Ø I1=1,25,1
      I2=I1**2
      WRITE(3,1Ø) I1,I2
3Ø    CONTINUE
      STOP
      END
```

The DO statement in this program works like this: the computer works down the program in its normal fashion, then when it executes the statement " DO 3Ø I1=1,25,1", it sets the variable I1 equal to 1. The statement "I2=I1**2" is executed, then statements below it until line number 3Ø is executed. After the computer has executed line 3Ø, it adds 1 to the variable I1, and then checks to see if I1 is greater than 25. If I1 is not greater than 25, the computer will go back and start working down from the statement after the DO, until it reaches the line numbered 3Ø, when it will make the same test again. When I1 is 25, after the computer executes statement 3Ø, it adds 1 to I1, making I1 equal 26. I1 is now greater than 25, so it will not jump back to the statement after the DO, but will just keep working down the program, executing the STOP statment.

In general, a DO statement has the form:

DO ### <ivar>=<initial>,<ifinal>,<istep>

is the number of the last statement of the segment of the program which is being looped. This statement can be any executable statement except a GOTO, IF, or another DO statement. The CONTINUE statement is often used for this purpose, as the computer does nothing when it executes it. For this reason, it is used to place the number "###" in the program. If control passes out of this segment of the program, the loop is discontinued.

In the general form, <ivar> must be an integer variable. When the DO statement is executed, <ivar> is set to the value of <initial>, which is an integer constant (like 1, as in the program SQUARES) or an integer variable (such as ICEPIK). <ifinal>, and <istep> can also be either integer

constants or variables. Each time the computer jumps from the bottom of the loop to the top, the value of <ivar> will be incremented, that is, the <istep> specified in the DO statement will be added to <ivar>. If the <istep> is not specified, a value of 1 will be used. This series of actions continues until <ivar> is greater than the value <ifinal> which was specified in the DO statement. When <ivar> is greater than <ifinal>, control is passed out of the loop, and normal execution resumes. After the loop is finished, <ivar> is undefined.

Some examples of DO statements used in programs will help make this a little clearer.

```
.LIST
COUNT1
10      FORMAT(I3)
        DO 20 I=2,20,2
        WRITE(3,10) I
20      CONTINUE
        STOP
        END
.EXECUTE
LOADING    FORTRAN 'COUNT1'

EXECUTION  FORTRAN 'COUNT1'


  2
  4
  6
  8
 10
 12
 14
 16
 18
 20


EXECUTION COMPLETED

.
```

If the DO statement is changed to be

```
DO 2Ø I=7,11
```

the program will print the following:

```
 7
 8
 9
 1Ø
 11
```

Using a DO loop, the following program finds how many positive integers must be summed to make a total greater than 1ØØ.

```
SUMMER
1Ø    FORMAT(' IT TAKES ',I2,' INTEGERS')
      ISUM=Ø
      DO 1ØØ J=1,1ØØ
      ISUM=ISUM+J
      IF (ISUM-1ØØ) 1ØØ,1ØØ,2ØØ
1ØØ   CONTINUE
      STOP
2ØØ   WRITE(3,1Ø) J
      STOP
      END
```

When the IF statement causes control to pass out of the loop, the loop is stopped.

```
.EXECUTE

LOADING   FORTRAN 'SUMMER'

EXECUTION FORTRAN 'SUMMER'


IT TAKES 14 INTEGERS

EXECUTION COMPLETED
.
```

Any time that a section of a program needs to be repeated a known number of times, a DO loop can be used. The following program averages grade points, and uses a DO loop to repeat a large portion of the program five times.

```
.LIST
GRADES
1Ø      FORMAT(F7.3,F7.3,F7.3,F7.3)
2Ø      FORMAT(' FOR STUDENT ',I2,' AVERAGE =',F7.3)
        DO 3Ø KOUNT=1,5
        READ(2,1Ø) GR1,GR2,GR3,GR4
        AVER=(GR1+GR2+GR3+GR4)/4.Ø
        WRITE(3,2Ø) KOUNT,AVER
3Ø      CONTINUE
        STOP
        END
ØØ3.ØØØØØ2.ØØØØØ3.ØØØØØ4.ØØØ
ØØ4.ØØØØØ4.ØØØØØ3.ØØØØØ4.ØØØ
ØØ3.ØØØØØ2.ØØØØØ3.ØØØØØ3.ØØØ
ØØ3.ØØØØØ2.ØØØØØ2.ØØØØØ1.ØØØ
.EXECUTE

LOADING    FORTRAN 'GRADES'

EXECUTION  FORTRAN 'GRADES'

 FOR STUDENT  1 AVERAGE =  3.ØØØ
 FOR STUDENT  2 AVERAGE =  3.75Ø
 FOR STUDENT  3 AVERAGE =  2.75Ø
 FOR STUDENT  4 AVERAGE =  2.ØØØ

        READ(2,1Ø) GR1,GR2,GR3,GR4
!!!!!! OUT OF PROGRAM DATA !!!!!!
FATAL ERROR-- EXECUTION TERMINATED

.
```

Possibly you noticed the unusual ending of this program. The error message means that there was not enough data for the program to read. When it ran out, it notified the user with the !!!!!! OUT OF PROGRAM DATA !!!!!! error message, and since this is a "FATAL ERROR", execution was halted. One way to correct this problem is to reload the program, changing the DO statement to " DO 3Ø KOUNT=1,4".

1) List the values the variable IVAL will go through as each of these statements is "looped". Also, tell what is the final line of the loop. For this exercise, IMP1=1, IMP2=25.

a) DO 2Ø IVAL=1,2Ø
b) DO 4Ø IVAL=IMP1,IMP2
c) DO 19Ø IVAL=1Ø,28,2
d) DO 77 IVAL=1ØØ,2ØØ,IMP2

2) What will each of the following programs do? Write the sample output for the sample input, if given.

a)
```
LOOP1
1Ø    FORMAT(' IDAHO:  FAMOUS POTATOES')
      DO 15 I=1,7,1
      WRITE(3,1Ø)
15    CONTINUE
      STOP
      END
```

b)
```
LOOP2
1ØØ   FORMAT(I3)
2ØØ   FORMAT(I3,' *',I3,'=',I6)
      READ(1,1ØØ)IST
      DO 3ØØ IMUL=1,1Ø
      IPROD=IST*IMUL
      WRITE(3,2ØØ) IST,IMUL,IPROD
3ØØ   CONTINUE
      STOP
      END
```

Sample input:ØØ4

3a) Change LOOP1 (problem 2a) to print out FAMOUS TATERS--IDAHO, six times.

b) Change LOOP2 (problem 2b) to print out the product of the inputted value of all even integers less than 25.

4) In mathematics, 12! is read twelve factorial. 12 factorial is defined as 1*2*3*4*5*6*7*8*9*1Ø*11*12. Write and run a program to find the factorial of an inputted value.

5) Write a program to print a table of the first 12 factorials.

6) Write a program to see if an inputted integer value is a perfect cube, using a DO loop.

As you have probably already discovered, errors are a common problem in computer programming. If you are just beginning to program, you can be reassured by the fact that even professional programmers have these same kinds of problems. But as your programs get longer, it takes more and more time to retype them every time you find an error. For this reason, OMSI 2ØØØ FORTRAN has a series of editing commands which allow the programmer to change his program one line at a time. These commands are all used from the monitor.

All OMSI 2ØØØ FORTRAN editing commands reference lines by special 'sequencing numbers'. The command LN will cause the monitor to list the current program with these numbers.

```
.LN
(  1)  SIMPLE
(  2)  1Ø     FORMAT (F1Ø.4)
(  3)         TOTAL=2.Ø+2.Ø
(  4)         WRITE (3,1Ø) TOTAL
(  5)         STOP
(  6)         END
.
```

The numbers enclosed in parantheses are the sequencing numbers. They are not part of the statement, they are just for convenience in editing. THEY ARE NOT LINE NUMBERS. Note that the name of the program is considered the first line.

The editing command that is the most useful in correcting your error-laden program is the CHANGE command. This command is used to replace a single line in the user's FORTRAN program with a new line, without altering the rest of the program. The general form of the CHANGE command is ###C-text. The '###' stands for the sequencing number of the line to be changed, and 'text' is what the line is to be changed to. For example, the program SIMPLE can be changed to add 3.Ø and 3.Ø by a single editing command, instead of reloading it.

```
.LN
(  1)  SIMPLE
(  2)  1Ø      FORMAT (F1Ø.4)
(  3)          TOTAL=2.Ø+5.Ø
(  4)          WRITE (3,1Ø) TOTAL
(  5)          STOP
(  6)          END
.3C-I^TOTAL=3.Ø+3.Ø
.LN
(  1)  SIMPLE
(  2)  1Ø      FORMAT (F1Ø.4)
(  3)          TOTAL=3.Ø+3.Ø
(  4)          WRITE (3,1Ø) TOTAL
(  5)          STOP
(  6)          END
.
```

Notice that the entire line was changed, so it was neccessary to use the control I to tab over to the seventh column. If the statement were to have a line number, it would have to be in the first six spaces of the new line.

The DELETE command allows the removal of single lines of text form the current FORTRAN program. It has the form ###D. It will delete the sequence line '###' from the program. The sequence number is always the number of the line, counting down from the name. When a line is deleted, the numbering on the rest of the program is changed. This should be kept in mind when using a series of editing commands. As an example of the DELETE command, an unwanted STOP statement can be removed from the following program.

```
.LN
(  1)  COW
(  2)  1Ø      FORMAT (' THE OREGON DAIRY COUNCIL')
(  3)  2Ø      FORMAT (' KEEPING THE COWS HAPPY!!')
(  4)          WRITE(3,1Ø)
(  5)          STOP
(  6)          WRITE(3,2Ø)
(  7)          STOP
(  8)          END
.5D
.LN
(  1)  COW
(  2)  1Ø      FORMAT (' THE OREGON DAIRY COUNCIL')
(  3)  2Ø      FORMAT (' KEEPING THE COWS HAPPY!!')
(  4)          WRITE(3,1Ø)
(  5)          WRITE(3,2Ø)
(  6)          STOP
(  7)          END
.
```

To insert lines in to the current program, there is the INSERT command. This allows the user to put addtional lines in the middle of his program, without reloading it. This command has the form ###I-text. Again '###' is a sequence number, here specifying that the 'text' is to be inserted as the line before '###'. INSERT also changes the sequence numbers of all the statements that follow it in the program. As an example of this command, it will be used to insert lines to have the following program print a heading.

```
.LN
(  1)  SQUARE
(  2)  1Ø     FORMAT (I3,I8)
(  3)         DO 3Ø I=1,1Ø
(  4)         ISQUAR=I**2
(  5)         WRITE(3,1Ø) I,ISQUAR
(  6)  3Ø     CONTINUE
(  7)         STOP
(  8)         END
.3I-2ØI^FORMAT (' TABLE OF SQUARES FROM 1 TO 1Ø')
.LN
(  1)  SQUARE
(  2)  1Ø     FORMAT (I3,I8)
(  3)  2Ø     FORMAT (' TABLE OF SQUARES FROM 1 TO 1Ø')
(  4)         DO 3Ø I=1,1Ø
(  5)         ISQUAR=I**2
(  6)         WRITE(3,1Ø) I,ISQUAR
(  7)  3Ø     CONTINUE
(  8)         STOP
(  9)         END
.4I-I^WRITE(3,2Ø)
.LN
(  1)  SQUARE
(  2)  1Ø     FORMAT (I3,I8)
(  3)  2Ø     FORMAT (' TABLE OF SQUARES FROM 1 TO 1Ø')
(  4)         WRITE(3,2Ø)
(  5)         DO 3Ø I=1,1Ø
(  6)         ISQUAR=I**2
(  7)         WRITE(3,1Ø) I,ISQUAR
(  8)  3Ø     CONTINUE
(  9)         STOP
( 1Ø)         END
.
```

The insert command allows you to insert text in before a line already in the program. To insert text at the end of the program, the APPEND command is used. It has form A-text. The text will be made the final line in the program. As an example, in the last section, the program GRADES needed an additional line of data to run correctly. This can be added with the append command.

```
.LN
(  1)  GRADES
(  2)  1Ø     FORMAT(F7.3,F7.3,F7.3,F7.3)
(  3)  2Ø     FORMAT(' FOR STUDENT ',I2,' AVERAGE =',F7.3)
(  4)         DO 3Ø KOUNT=1,5
(  5)         READ(2,1Ø)GR1,GR2,GR3,GR4
(  6)         AVER=(GR1+GR2+GR3+GR4)/4.Ø
(  7)         WRITE(3,2Ø) KOUNT,AVER
(  8)  3Ø     CONTINUE
(  9)         STOP
( 1Ø)         END
( 11)  ØØ3.ØØØØØ2.ØØØØØ3.ØØØØØ4.ØØØ
( 12)  ØØ4.ØØØØØ4.ØØØØØ3.ØØØØØ4.ØØØ
( 13)  ØØ3.ØØØØØ2.ØØØØØ3.ØØØØØ3.ØØØ
( 14)  ØØ3.ØØØØØ2.ØØØØØ2.ØØØØØ1.ØØØ
.A-ØØ3.ØØØØØ4.ØØØØØ2.ØØØØØ3.ØØØ
.LN
(  1)  GRADES
(  2)  1Ø     FORMAT(F7.3,F7.3,F7.3,F7.3)
(  3)  2Ø     FORMAT(' FOR STUDENT ',I2,' AVERAGE =',F7.3)
(  4)         DO 3Ø KOUNT=1,5
(  5)         READ(2,1Ø)GR1,GR2,GR3,GR4
(  6)         AVER=(GR1+GR2+GR3+GR4)/4.Ø
(  7)         WRITE(3,2Ø) KOUNT,AVER
(  8)  3Ø     CONTINUE
(  9)         STOP
( 1Ø)         END
( 11)  ØØ3.ØØØØØ2.ØØØØØ3.ØØØØØ4.ØØØ
( 12)  ØØ4.ØØØØØ4.ØØØØØ3.ØØØØØ4.ØØØ
( 13)  ØØ3.ØØØØØ2.ØØØØØ3.ØØØØØ3.ØØØ
( 14)  ØØ3.ØØØØØ2.ØØØØØ2.ØØØØØ1.ØØØ
( 15)  ØØ3.ØØØØØ4.ØØØØØ2.ØØØØØ3.ØØØ
.
```

The program now has the correct amount of data and will run without the 'OUT OF PROGRAM DATA' error. This is the general purpose of these commands, to allow fast and easy debugging. And if you are not careful, they let you make your mistakes faster too.

So far, three format specificaions, "F", "I", and "'", have been introduced. There are three more, "E", "X", and "/", available. There is also an easy way to repeat a single specification.

To generate linefeeds (move the paper up one line), the "/" is used. A / can be placed at the first of FORMAT line - FORMAT(//' THIS IS A TEST') - or at the end of a FORMAT line - FORMAT(' THIS IS ANOTHER TEST'///) - or between other specifcations in place of a comma - FORMAT(//I3//I5/). To show how this works, try running this program

```
LINEFEEDER
1Ø    FORMAT(//' LINEFEEDS'//' MORE LINEFEEDS'//)
      WRITE (3,1Ø)
      STOP
      END
```

X formatting is simple. If an "X" is found in a FORMAT statement, a blank space is printed. For example, 1Ø FORMAT(I4,X,I3) used to print 1234 and 567 would look like "1234 567". How to get more than one space at a time will be explained a little later.

E formatting is more complicated. It is used with floating point values and prints them out in scientific notation, 234567.8 would be written as "2.345678E+Ø5". The expontent is Ø5, indicating that the whole value should be multiplied by ten to fifth power. The general form of the E specification is "Ew.d". "w" is the width of the field in which the number will be printed and "d" is the number of digits after the decimal point. "E12.4" will reserve a field twelve spaces wide, with four digits after the decimal point (###.####E+##). To show this more clearly, SIMPLE can be modifed to use E format.

```
SIMPLE7
1Ø    FORMAT(E12.5)
      TOTAL=2.Ø+5.Ø
      WRITE(3,1Ø)TOTAL
      STOP
      END

.EXEC

LOADING   FORTRAN 'SIMPLE7'

EXECUTION FORTRAN 'SIMPLE7'

+7.ØØØØØE+ØØ

EXECUTION COMPLETED

.
```

E formatting is useful for outputting extremely large numbers. When using the E specification, space must be reserved for the exponent as well as the decimal point and the sign.

To repeat an I, F, X, or E specification, an integer (called a replicator) is placed in front of the letter and then this specification will be repeated that many times. "4I3" is the same as "I3,I3,I3,I3". "1Ø FORMAT(2F5.2,3X,3I6)" will cause the same output as "1Ø FORMAT(F5.2,F5.2,X,X,X,I6,I6,I6)". The use of a replicator allows long strings of specifications to be written in a much shorter manner.

The 'literal field' and X formatting are primarily for ouput, but they can occur in a FORMAT statement referenced by a READ. If they do, the field they cover is skipped over and no values are READ from it. To clarify this, suppose these statements were part of a program

```
1Ø    FORMAT(I3,X,I4)
       ---
      READ(2,1Ø)IN1,IN2
       ---
      END
12345678
```

When the READ statement is executed, it will read IN1 from the first three spaces of the data line, then skip one space, and read IN2 from spaces five through eight. This will set IN1 to 123 and IN2 to 5678. The same thing would have occurred if there had been a one character long literal field instead of the X.

Formatting output is something of an art, and it requires practice. A program is worthless if its output cannot be read.

S E C T I O N T W O

PROGRAMMING PROBLEMS

Programming Problems

All of the problems in this section can be solved with programs using only the parts of FORTRAN presented in the first section; however, many of them can be solved faster and easier using some of the additional parts of FORTRAN explained in the third section (Additional Features of OMSI 2ØØØ FORTRAN).

1) Write a FORTRAN program to find the greatest common factor of two two-digit integers. Values should be read from the teleprinter.

2) Write a program to reduce fractions to their simplest form. That is, if Ø2 was input as the numerator and Ø4 input as the demoninator, the computer would print out " 2/ 4= 1/ 2"

3) Write a program to add two fractions, reduce their sum, and print the result out in fractional form. Sample output might be " 1/ 6+ 1/ 2= 2/ 3"

4) Write a program to test if a number is prime.

5) Write a program to find the first 25 prime numbers.

6) Earning 5% interest compounded quarterly, how long will it take for a 2Ø dollar investment to grow to over 1ØØ dollars if all interest is reinvested.

7) The tire on a 1971 gray-green Ford Pinto has a diameter of 16.5 inches. How many times will this tire revolve on a 4Ø mile trip?

8) Write a program to find the least common multiple of two input integers.

9) Write a program to convert a number of seconds into a number of days, hours, minutes, and seconds. For example 1ØØØØØ seconds= 1 day 3 hours 46 minutes and 4Ø seconds.

1Ø) The value of a new car decreases by about 2Ø% per year. Design and run a program that will print a table of the car's value every year between the year of purchase and 1975.

11) Binary is base two. All numbers in binary have only zeroes and ones for digits. Write a FORTRAN program to convert any 4 digit positive integer to binary.

12) Using the equation PV=nRT, where
P=pressure in torr
V=volume in liters
n=number of molecules, in moles
R=62.4 torr-liter/mole-degree (the gas constant)
and T=temperature in degrees absolute
write a FORTRAN program that find the change in pressure as the temperature goes from 27Ø to 35Ø degrees in steps of 1Ø. Assume a 3Ø liter volume and 1.5 moles of gas molecules.

13) Using the formula
$2AgNO_3 + CaCl_2 \rightarrow Ca^{++} + 2NO_3^- + 2AgCl$
find the amount of AgCl produced for a given amount of AgNO3 and CaCl2. All amounts are in moles. Assume the proportionally smaller amount will limit the yield.

14) Given that the acceleration due to gravity is -1Ø.Ø meters/sec-sec, print a table of the heighth of a projectile fired straight up at an input velocity. Use two second intervals, and stop the table when the projectile strikes the ground. Use the formula $d=v_Ø t+\frac{1}{2}at^2$.

15) Inputting the angle from horizontal and the velocity determine the distance from the target the projectile will strike the ground.

16) Write a cash register program. The clerk using the register gets a 1Ø% commission for all sales. She uses the following codes

Ø = Stop and print out total sales and her commissions.
1 = Write up a sale
2 = Write up a refund

She gets commissions on all sales and she is not docked for refunds. After she types 1 or 2 she will input the value of the sale or refund.

17) Given the time of day on a 24 hour clock and the number of time zones away a given city is, print out the time in that city. Include + or - 1 day, if appropriate. Ignore the international date line.

18) The square root of a number S can be approximated using the following formula

$$X_{i+1} = \frac{X_i^2 + S}{2X_i}$$

Where X is an approximation of the square root of S. Write a FORTRAN program to find the square root of an input value to within .Ø1% error. Do not use the SQRT function.

19) The sine of an angle can be found using the series

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots,$$

where x is an angle in radians. Remembering that $\sin(x)^2 + \cos(x)^2 = 1$, approximate the sine and cosine of the angles Ø, 1Ø, 2Ø, 3Ø, 4Ø, 5Ø, 6Ø, 7Ø, 8Ø, and 9Ø degrees.

2Ø) Write a FORTRAN program to solve first degree simultaneous equations of two variables.

S E C T I O N T H R E E

MORE ABOUT FORTRAN

for the 2ØØØC/F version

The parts of the FORTRAN language discussed in this section are available only on the 2ØØØC/F version of OMSI 2ØØØ FORTRAN. If you are not certain which version you are on, check to see if the monitor prints a question mark after every period. If it does, you are using the 2ØØØB/E version and the statements described in this section are not available. If there is not a question mark after every period, you are the 2ØØØC/F version and this section contains information that you will find handy in writing FORTRAN programs.

LOGICAL IF STATEMENTS

The IF statements discussed earlier in this book were always of the form IF(<exp>)<line#>,<line#>,<line#>. These are called "multibranch IF" statements. OMSI 2ØØØC/F FORTRAN has another type of IF statement called the "logical IF." All logical IF's have the form

IF(<logical expression>)<executable statement>

For example:

```
IF(ALPHA)GOTO 1Ø
IF(ALPHA-1Ø.Ø)WRITE(3,1ØØ)ALPHA,BETA
```

The logical IF bases its decision on the "truth" of the expression. If the expression is "true", then the executable statement is executed, and then the program continues. If the expression is "false"the executable statement is ignored and the the next line is executed. The computer decides whether the expression is true or not depending on its value. If the value is zero, the expression is false. If the value is not zero, it "true".

Using logical IFs can make a program much shorter as well as increasing the effeciency of the program. The program TENPOWER uses multibranch IFs to tell whether a number is a multiple of ten or not.

```
TENPOWER
1ØØ    FORMAT(I5)
2ØØ    FORMAT(I5,'# IS ')
3ØØ    FORMAT('#NOT ')
4ØØ    FORMAT(' A MULTIPLE OF TEN')
35Ø    READ(1,1ØØ)IVAL
       IF(IVAL)375,9ØØ,375
375    WRITE(3,2ØØ)IVAL
       IF(IVAL-IVAL/1Ø*1Ø)45Ø,5ØØ,45Ø
45Ø    WRITE(3,3ØØ)
5ØØ    WRITE(3,4ØØ)
       GOTO 35Ø
9ØØ    STOP
       END
```

TENPOWER could easily be rewritten to use logical IF statements to yield the same results. As you read the program note the use of carriage control.

```
TENPOWER
1ØØ    FORMAT(I4)
2ØØ    FORMAT(I4,'# IS ')
3ØØ    FORMAT('#NOT ')
4ØØ    FORMAT(' A MULTIPLE OF TEN')
35Ø    READ(1,1ØØ)IVAL
       IF(IVAL)GOTO 375
       STOP
375    WRITE(3,2ØØ)IVAL
       IF(IVAL-IVAL/1Ø*1Ø)WRITE(3,3ØØ)
       WRITE(3,4ØØ)
       GOTO 35Ø
       END
```

Both versions print out the same results. The logical IF is just more convient.

To go with the logical IF statment, a set of logical and relational operators have been included in te C/F version. The relational operators are used to compare two quantities. The available relational operators are:

.EQ.	(equal to)
.NE.	(not equal to)
.LT.	(less than)
.LE.	(less than or equal to)
.GT.	(greater than)
.GE.	(greater than or equal to)

The periods are part of the operator. To use them, they are inserted between the two quantities you want to compare. If the relation holds, it is true, and the entire expression is given the value 1. If the relation does not hold, it is false and given the value zero. For example, "1Ø.Ø.EQ.12.Ø" is false so it would be given the value zero. In a logical IF, such as

```
IF(1Ø.Ø.EQ.12.Ø)WRITE(3,1ØØ)
```

the WRITE(3,1ØØ) would not be done because the expression is false. Logical operators are used to logically combine two values, that is test the "truth" of two values and determine the truth of the entire expression depending on the relation between the "truth" of the two values. The logical operators available are

.AND.	(both values must be true)
.OR.	(either of the values is true)
.EOR.	(one or other value is true but not both)

If the logical relation holds, the expression is given a value of 1 ("true"); if the relationship does not hold, the expression is given a value of Ø ("false"). For example:

```
1.Ø.AND.2.Ø = 1.Ø (TRUE)
1.Ø.AND.Ø.Ø = Ø.Ø (FALSE)
1.Ø.OR.Ø.Ø  = 1.Ø (TRUE)
1.Ø.EOR.Ø.Ø = 1.Ø (TRUE)
1.Ø.EOR.1.Ø = Ø.Ø (FALSE)
```

These operators are most useful in the logical IF. Suppose a program required that if the variable ALK was less than zero and the variable TLK was less than or equal to the variable SWIPE then all three values were to be printed out, the following logical IF could be used

```
IF((TLK.LT.Ø.Ø).AND.(TLK.LE.SWIPE))WRITE(3,1Ø1)ALK,TLK,SWIPE
```

These operators will work in either mode (real or integer) as long as the mode is consistent through out the expression. These operators are evaluated left to right, that is, without any parantheses, the left most logical or relational operator will be done first and then the next one across, until the end of the expression.

SUBSCRIPTED VARIABLES

OMSI 2ØØØC/F FORTRAN allows the use of subscripted variables in assignment, IF, READ, and WRITE statements. To use subscripted variables, a variable must be dimensioned to be an array of values. If the variable ARRAY was dimensioned to 1Ø, the computer would reserve space for ten values under the name ARRAY. The first value would be referenced by ARRAY(1), the second by ARRAY(2), and so on up to ARRAY(1Ø). The importance of subscripting is that it allows programs to be shorter and more effecient. Suppose a program was needed to see if any two of ten inputted vaues were equal. Without the use on an array this program would be extremely long. With an array, the search it self can be five lines long. If the ten values were stored in an array called VALUE, the search segment might look like this

```
      DO 1ØØ I=1,1Ø
      I1=I-1
      DO 1ØØ J=1,I1
      IF(VALUE(I).EQ.VALUE(J))GOTO 2ØØ
1ØØ   CONTINUE
```

Without an array, this segment would have needed over 5Ø seperate IF statements.

To use the array, it must be dimensioned. For this purpose there is the DIMENSION statement. A DIMENSION statement is a non-executable statement and must be the very first non-executable statement in the program. Since non-executable statements must be at the first of a program, DIMENSION, if it is used, must be the very first statement in the program. In the DIMENSION statement, the names of the arrays and the number of values they are to have must be specified. In OMSI 2ØØØ FORTRAN, a maximum of twelve seperate arrays may be dimimensioned, with a maximum of two hundred seperate values. Arrays can have one, two, or three subscripts. The rules that determine te mode of an array are the same as the rules for the modes of non-subscripted variables. Once a variable is used an array name, it cannot be used as a non-subscripted variable. Some sample DIMENSION statements are

```
DIMENSION VALUE(1Ø)
DIMENSION ARRAY(1Ø),ICZ(1Ø,1Ø)
DIMENSION LUGNUT(4,4,4),SPEEDO(6,6),IMP(4Ø)
```

To show a litle more about how arrays are used, the following program inputs ten values and tells which are equal.

```
EQUAL
      DIMENSION VALUE(1Ø),ISAME(6Ø,2)
1Ø    FORMAT(F1Ø.4)
2Ø    FORMAT(' VALUE #',I2,' AND VALUE #',I2,' ARE EQUAL')
      DO 5Ø I=1,1Ø
5Ø    READ(1,1Ø) VALUE(I)
      K=Ø
      DO 1ØØ I=1,1Ø
      I1=I-1
      DO 1ØØ J=1,I1
      IF(VALUE(I).NE.VALUE(J))GOTO 1ØØ
      K=K+1
      ISAME(K,1)=I
      ISAME(K,2)=J
1ØØ   CONTINUE
      DO 2ØØ K1=1,K
      WRITE(3,2Ø)ISAME(K1,1),ISAME(K1,2)
2ØØ   CONTINUE
      STOP
      END
```

When using subscripts and arrays, there are three important rules to remember. ALL ARRAYS MUST BE DIMENSIONED. If an array is not dimension, and the variable later appears in the program with a subscript, the program will not run. The second important rule to remember is SUBSCRIPTS MUST NOT EXCEED THE DIMENSIONED SIZE. Again if this happens the program will not run. And thirdly, A SUBSCRIPTED VARIABLE MUST HAVE THE SAME NUMBER OF SUBSCRIPTS AS IT WAS DIMENSONED. Using these rules, subscripted variables can make many programs shorter and more effecient.

OMSI 2000 B/E FORTRAN SUMMARY

COMMANDS

Command	Purpose
LOAD	Accept a FORTRAN program
AI (ctrl I)	Tab over line number field
AY (ctrl Y)	Exit from load command
EXECUTE	Executes FORTRAN program
LIST	Print text of program
TPUNCH	Punches tape of program
TAPE	Accept program from tape
KJOB	Returns to BASIC, deleting current program
UJOB	Returns to BASIC, without deleting program
HELP	Summary of system operation

EDITING COMMANDS

LN	Lists program with sequence numbers
###C-text	Changes sequenced numbered line to text
###I-text	Inserts "text" before line ###
###D	Deletes line ###
A-text	Makes "text" final line of the program

CATALOG COMMANDS

CA	Gives catalog of programs library file
ST	Stores current program in library file
R name	Retrieves program "name" from library file
E name	Erases program "name"

FUNCTIONS

REAL argument, REAL value

SIN(rx)	sine of rx in radians
COS(rx)	cosine of rx
SQRT(rx)	Square root
ABS(rx)	absolute value
EXP(rx)	e to the power rx
AINT(rx)	truncation
ATAN(rx)	arctan in radians
ALOG(rx)	log base e
ALOG10(rx)	log base 10

INTEGER argument, INTEGER value

IABS(ix)	absolute value

MODE CHANGING

FLOAT(ix)	integer to real
IFIX(rx)	real to integer
INT(rx)	real to integer

STATEMENTS

Statement form	Remarks
<var>=<exp>	Stores value of <exp> in <var>
C comment	A "C" in the first column causes rest of the line to be ignored.
CONTINUE	No operation, dummy statement.
DO ### <ivar>=<initial>,<ifinal>,<istepp>	Causes lines down to and including ### to be repeated 1+(<ifinal>-<initiail>)/<istep> times. <istep> default is +1.
END	Indicates end of text.
FORMAT(<spec>,<spec>,...)	Formats input and output. Specifications allowed are: nIw - Integer; nEw.d - Floating point; nEw.d - Expontent form; nX - Spacing; 'text' - Literal field where "n" is the replicator, "w" is the field width, "d" is the number of digits after the decimal point, and "text" is a string.
GOTO ###	Transfer control to line with line number ###
IF (<exp>) ##1,##2,##3	Branches depending on the value of <exp> to ##1, if <exp> < 0; to ##2, if <exp> = 0; to ##3, if <exp> > 0
PAUSE <number>	Causes 5 second delay in execution. <number> will be printed out, if present.
READ (1,###) <var list>	Reads values for variables from the terminal, using FORMAT statement ###.
READ (2,###) <var list>	Reads values for variables from data, using FORMAT statement ###. Everything after the END is data.
STOP	Halts execution.
Write (3,###) <var list>	Prints values of variables out on the terminal using FORMAT statement ###.

Variables and operations

Variables may have 1 to 6 characters. The first must be a letter, other may be letters or digits. If the first letter is a I, J, K, L, M, or N the variable will be an integer. Starting with any other first letter will make the variable floating point. All operations must take place between two quantities of the same mode.

OPERATORS

**	exponentiation
*	multiplication
/	division
+	addition
-	subtraction

Printed in USA